REMARKS

Claims 1, 6, 11-12, and 15 have been amended. Claims 4, 9, and 13 have been canceled. No new claims have been added. Claims 1-3, 5-8, 10-12, and 14-15 are pending.

Claim 1-3, 5-8, 10-12, and 14-15 stand rejected under 35 U.S.C. 102(e) as being anticipated by Abbott (U.S. Publication 2003/0033344). This rejection is respectfully traversed.

Claims 1, 6, 11, and 15 recite, *inter alia,* "constructing a continuation block, said continuation block comprising a block header and a stack fragment, said stack fragment comprising a range of said stack memory between a current stack top address and a current stack base address; and pushing said continuation block onto said stack memory;" and "popping said continuation block from said stack memory; pushing said stack fragment portion of said continuation block onto said stack memory; setting said current stack top address to a start of said stack fragment of said stack memory; and setting said current stack base to a end of said stack fragment of said stack memory."

Claim 12 recites a similar limitation in mean-plus-function format.

Abbot is directed to an improvement in the implementation of a Java virtual machine (JVM). Java applications are executed within the context of a JVM. Ordinarily, at start-up, each Java application assumes that the JVM is in a certain well defined "clean" state. That is, a new JVM is started for each Java application. Paragraph [0005].

Abbot discloses a JVM which includes the capability for saving and restoring "snapshots" of the entire state of the JVM. Such an ability would permit, for example, a running Java application to be suspending during its execution and then subsequently restored. The snapshot would include both the state of the Java application itself, as well as the state of its associated JVM. The save and restore functions may be implemented via respective APIs, and the APIs may be visible to a Java application, the operating system, or both. For example, a Java application wishing to suspend itself may call the save snapshot API, or alternatively, a power saving module (not within the JVM) wanting to hibernate the computer system may call the save snapshot API. Paragraph [0026]-[0027].

Fig. 7 illustrates the save snapshot process. The save snapshot process saves the entire state of the JVM. Some pre-processing is performed at steps 705-715 to ensure that the saved snapshot can be used during the restoration if, for example, the JVM itself is loaded by the operating system during the restore to a different memory addresses. Once the pre-processing is performed, the save snapshot process include steps for saving class information (step 720), heap information and the vector table (step 725), threads and stack information (730), DLL information (step 735), GUI and other I/O information (step 740), and for causing Java system components to perform their own save routines (step 745). The save snapshot process ends at step 750, at which point control can either be returned to the Java application for post snapshot processing, or the JVM (and thus also the application) can be simply terminated. Paragraph [0131]. The restore process is illustrated by Fig. 8, and restores the state of DLLs, heap/classes, threads/stacks, and I/O. At step 860, the restoration is complete and the Java application is permitted to resume execution.

In summary, Abbot discloses apparatus and method which performs saves and restores of the entire state of a JVM. Significantly, the restore process disclosed by Abbot returns the state of the JVM to the <u>identical state</u> as the JVM had at the end of the save process (e.g., step 750). Further, Abbot discloses saving the state to a <u>file</u>.

In contrast, the above referenced portions of independent claims 1, 6, 11-12, and 15 each recite apparatus, method, or storage medium instruction which when executed, include a save feature which constructs a continuation block to include one specific stack fragment, and which pushes the continuation block onto the stack as part of the save feature. Further, a restore feature is claimed which restores a <u>portion</u> (i.e., only the stack fragment portion is restored, the block header is not restored to the stack) of the previously stored stack fragment. This is illustrated in the application at Fig. 3, drawing element 304 (showing the stack at the end of the save) and Fig. 4, drawing element 404 (showing the stack at the end of the restore). Finally, the save/restore of the claimed invention is performed via pushing and popping a stack memory, while Abbot discloses storing the information upon a file.

Accordingly, Abbot fails to disclose or suggest the above referenced features of the independent claims.

Claims 1, 6, 11-12, and 15 are believed to be allowable over the prior art of record. The depending claims are also believed to be allowable for at least the same reasons as the independent claims.

Applicants believe that the present application is now in condition for allowance, which prompt and favorable action is respectfully requested.

CONCLUSION

In light of the amendments contained herein, Applicants submit that the application is in condition for allowance, for which early action is requested.

Please charge any fees or overpayments that may be due with this response to Deposit Account No. 17-0026.

Respectfully submitted,

Dated: February 27, 2006 By: ______________________

Christopher S. Chow
Reg. No. 46,493
(858) 845-3249

QUALCOMM Incorporated
Attn: Patent Department
5775 Morehouse Drive
San Diego, California 92121-1714
Telephone: (858) 658-5787
Facsimile: (858) 658-2502